

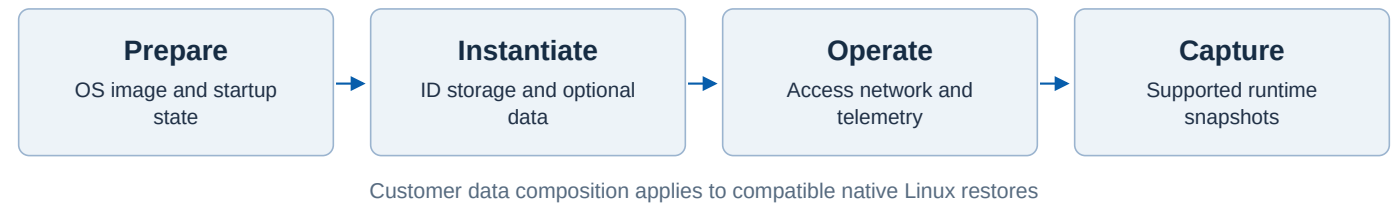
MYKOS key innovations

A technical leadership overview

Released design baseline: MYKOS 1.5.20261024-146

MYKOS brings prepared operating systems, customer application data, instance identity, networking, and controlled operations into one management model. Its central contribution is making those pieces work together as repeatable, ID-specific environments across native Linux containers and VM-backed operating systems.

This design is useful when a team needs to recreate a customer environment, investigate a problem, preserve working state, or operate several OS families without building a separate administration process for each one. MYKOS builds on Podman, QEMU/KVM, CRIU, Libreswan, Apache, FastAPI, and systemd; its design contributions are the lifecycle contracts and operational workflows that connect them.



The six design contributions

Design contribution	Operational value
A shared model across different OS engines	Teams manage images, ID-specific instances, and snapshots through common web, API, and command-line surfaces.
Reusable installed state	Prepared VM images can include matching disk, memory, and device state, reducing repeated installation and boot preparation.
Customer backups composed into a runtime	Compatible native Linux environments can use customer application trees while keeping instance identity and host configuration explicit.
Selectable VPN participation	Direct tunnels and shared gateways support different connectivity needs through explicit routing and policy choices.
Host-side observation and controlled operations	Collected state supports the dashboard and API, while authorization and privileged helpers govern changes.
Delivery separated from customer state	Signed packages and managed-file installation allow control-plane updates while retaining guest storage, identities, and operator settings.

A shared model across different OS engines

MYKOS gives native Linux containers and QEMU-backed Windows, Windows Server, macOS, and Ubuntu environments the same high-level lifecycle vocabulary. An image is a prepared OS and its required startup state. A container is an ID-specific running instance with its own operational identity. An intentional snapshot preserves instance state for a later restore.

The implementation retains OS-specific mechanisms beneath that model. Native Linux uses Podman processes and filesystems; VM-backed environments use QEMU and persistent guest storage. Console access follows the appropriate path: native PTYs, Ubuntu serial consoles, Windows guest-agent commands, or macOS SSH and graphical access.

For technical leadership, the value is a common operating contract. Automation and support procedures can organize work around environment identity and lifecycle while retaining the capabilities and constraints of each OS engine.

Reusable installed state

A prepared VM image can contain more than an installed disk. MYKOS captures matching memory and device state, flushes the corresponding disks, and records a manifest describing the capture. Creating an instance copies the prepared storage, requests a one-time restore when appropriate, and checks that restoration completed.

This shifts repeated work into image preparation. Installation, provisioning, and selected startup work can be completed once and reused for subsequent instances. Captures remain tied to compatible CPU, QEMU, memory, and host conditions; they are operational artifacts with explicit prerequisites.

MYKOS also separates startup captures from intentional snapshots. Startup captures serve image and instance launch. Intentional snapshots represent operator decisions and remain protected from automatic startup-capture pruning. That distinction gives teams a clearer retention and recovery model.

Customer backups composed into a runtime

For compatible native Linux restores, MYKOS combines a prepared OS with customer application trees from an existing backup set. Available paths can include `/etc`, `/home`, `/usr2`, `/d`, and `/usr/java`. The data is mounted into the instance, avoiding a mandatory full copy before the environment can start.

The difficult part is identity reconciliation. MYKOS preserves customer numeric account identities and relocates conflicting image accounts within the instance. It prepares private account files and machine-specific settings before startup, including hostname, machine ID, hosts entries, and selected boot and PAM configuration. This lets customer context and the base OS coexist without treating their configuration as interchangeable.

The operational benefit is a more faithful support environment with fewer manual reconstruction steps. These mounts are writable live storage. Teams that need an unchanged recovery source should provide a separate working copy and coordinate protection for application data and external writers.

Selectable VPN participation

MYKOS combines ID-based environment management with explicit network participation. Its managed private bridge provides per-instance addresses and deterministic MAC identities. Connection setup can use independent guest tunnels or a selected shared gateway, with the gateway hosted by the MYKOS host or a native Linux guest.

Shared routing can give different guest types access through a common VPN path, reducing the need to configure a tunnel inside each VM operating system. Host participation is also an explicit policy choice. Saved gateway profiles, routing ownership checks, boot guards, and reconciliation keep the intended topology visible and recoverable.

The implementation rejects conflicting shared-default and independent-tunnel scenarios. Shared routing includes scoped fail-closed protections and checks for the expected tunnel state. Correct operation still depends on the selected peer, selectors, return routes, and DNS path. Those details belong in an architecture and validation plan for each deployment.

Host-side observation and controlled operations

MYKOS separates ordinary observation from actions that change an environment. Host-side collectors inspect Podman metadata, resource statistics, files, and control sockets, then publish timestamped state for the web application and API. Normal host-monitoring code avoids executing guest programs during polling. Stale or unavailable observations can remain unknown instead of being interpreted as healthy.

Changes pass through authenticated interfaces and privileged helpers. The web application runs as an unprivileged service; root helpers validate callers, targets, and operation arguments through systemd-managed sockets. Browser sessions and scoped API keys are separate identities, and interactive terminals have their own authorization requirements. Explicit administrator command execution is a separate, powerful workflow.

This structure supports both operators and automation while making trust boundaries inspectable. Native containers are privileged, and the web service is a trusted control-plane component. Storage and network separation therefore operate within a trusted administrative host model.

Delivery separated from customer state

MYKOS packages its control plane independently of guest disks, snapshots, browser accounts, API credentials, and local configuration. Signed RH8, RH9, and RH10 RPMs provide a consistent delivery path. Managed-file manifests, rollback snapshots, lifecycle locks, and reference checks support deliberate updates and cleanup.

Operator customizations are also treated as inputs. Generated build copies adapt maintained kickstarts without replacing the originals. Image removal checks references and active use; existing instance storage is retained when an operation cannot safely finish.

The installer applies the same principle to adoption. Fresh installation provides IP-based application access and prints the URL after a readiness check. Domain and Certbot setup are enabled explicitly with `--configure-https`. This separates getting the application running from adopting a particular public-domain and certificate configuration.

The combined operating workflow

1. Prepare and validate an OS image suitable for the required workload.
2. Create an ID-specific instance with its own storage and identity.
3. Attach compatible customer data when needed, using a working copy when preservation is required.
4. Select the intended connectivity model and verify its traffic paths.
5. Observe and operate the environment through the web interface, CLI, or API.
6. Capture supported runtime state deliberately and coordinate protection for mounted application data.

The value comes from the connections between these steps. Image preparation becomes reusable. Customer context becomes an explicit composition input. Networking becomes a managed policy. Observation and automation share an operational model. Updates can be evaluated separately from the lifetime of customer environments.

How technical leaders can evaluate the value

Outcome to evaluate	Useful measurement
Faster environment preparation	Time from a request to a usable environment, including data and network validation.
Less repeated installation and copying	Installation steps repeated per case, bytes copied, and image or restore preparation time.
More consistent operations	Manual steps and configuration differences across OS families and repeated support cases.
Lower operational interference	Guest executions caused by monitoring, host collection cost, and freshness of reported state.
More predictable updates and recovery	Preserved guest identities and start times across updates, plus tested rollback and restore outcomes.

These measures turn the design into an assessable operating capability. Compatibility tests and deployment-specific results establish the performance, recovery, and scale characteristics for a particular environment.

Implementation foundations

Area	Principal implementation components
VM images and paired state	mykos_vm.py, mykos_image_build.py
Intentional snapshots	mykos_snapshots.py, mykos_criu.py
Backup composition and account reconciliation	mykos-stage-pos, mykos_backup_restore.py, mykos_backup_identity.py
Network identity and VPN policy	mykos_network.py, mykos_vpn_gateway.py, mykos_vpn_host.py
Observation and controlled actions	mykos_host_monitoring.py, mykos_runtime.py, systemd socket units
Delivery and local state preservation	mykos-install, mykos-bootstrap, RPM specification